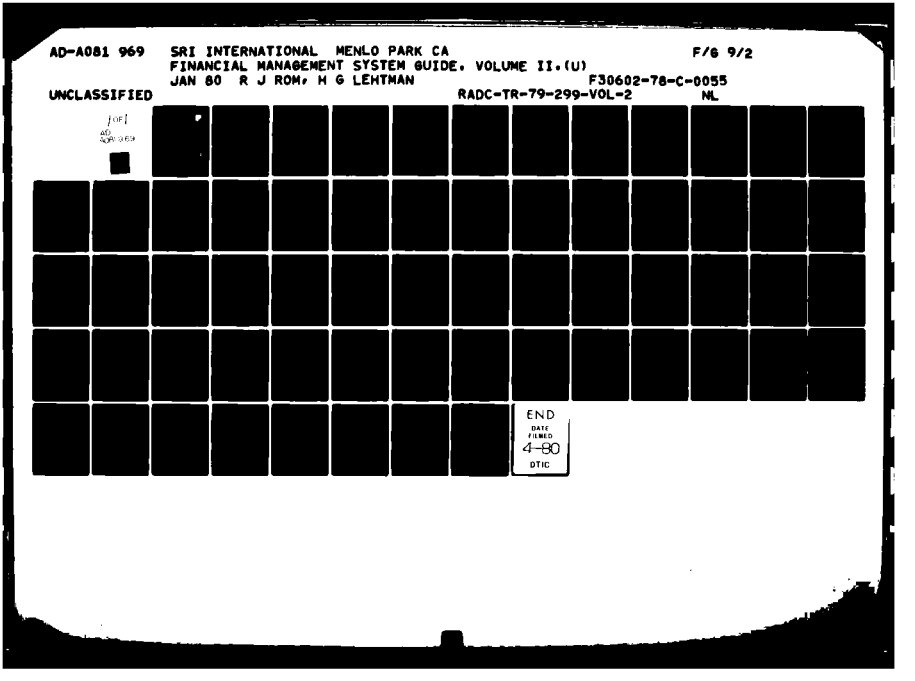
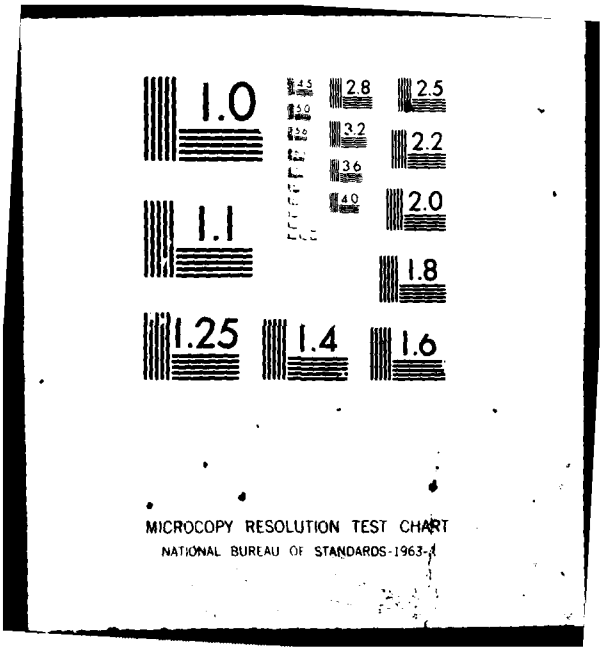

AD-A081 969
SRI INTERNATIONAL MENLO PARK CA
F/G 9/2
FINANCIAL MANAGEMENT SYSTEM GUIDE, VOLUME II.(U)
JAN 80 R J ROM, H G LEHTMAN
F30602-78-C-0055
UNCLASSIFIED
RADC-TR-79-299-VOL-2
NL
1 OF 1
AD
A081969
END
DATE FILMED
4-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RADC-TR-79-299, Vol II (of two)
Final Technical Report
January 1980

FINANCIAL MANAGEMENT SYSTEM GUIDE

SRI International

Raphael J. Rom
Harvey G. Lehtman

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-299, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Richard Calicchia

RICHARD CALICCHIA
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-299, Vol II (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): FINANCIAL MANAGEMENT SYSTEM GUIDE • Volume II	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report 10 Jan 78 - 9 Aug 79
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): Raphael J. Rom, Harvey G. Lehtman	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0055
9. PERFORMING ORGANIZATION NAME AND ADDRESS: SRI International, 333 Ravenswood Avenue, Menlo Park CA 94025	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55812006, 20
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441	12. REPORT DATE: Jan 80; NUMBER OF PAGES: 62
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (19) TR-79-299-VOL-2	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

12) 62

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Richard Calicchia (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

On-line
software engineering
programming environments
computers
system software
FMS
augment

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes how the Financial Management System (FMS) and its associated Data Entry System (DES) work from a technical point of view. It is intended as an aid to system programmers who may be interested in modifying the existing system in the future. We describe the files involved in the use of FMS/DES and internal procedures that may be of use to those wishing to extend the features of the system through the creation of user utility programs.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20 (Cont'd)

The first two sections of this document outline the structure of the FMS data base. Descriptions are given of the files that make up the data base, the basic elements (fields and records) that make up files, and special features supplementing the basic AUGMENT file system to support file and system integrity.

The third section describes basic system algorithms for creation, modification, and display of the data base elements. The fourth section describes the structure of the Data Dictionary, which determines the actual format of particular data elements. The fifth section describes other data structures used by the system. The appendix lists the names of system procedures grouped by function.

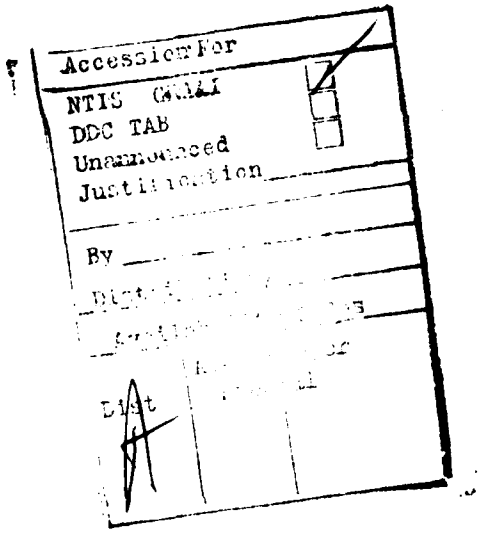


TABLE OF CONTENTS

INTRODUCTION

This document describes how the Financial Management System (FMS) and its associated Data Entry System (DES) work from a technical point of view. It is intended as an aid to system programmers who may be interested in modifying the existing system in the future. We describe the files involved in the use of FMS/DES and internal procedures that may be of use to those wishing to extend the features of the system through the creation of user utility programs. We assume the reader is familiar with FMS/DES as a user, and is acquainted with the AUGMENT system and its associated programming language, L10. (Other documents are available that describe these subjects.)

The first two sections of this document outline the structure of the FMS data base. Descriptions are given of the files that make up the data base, the basic elements (fields and records) that make up files, and special features supplementing the basic AUGMENT file system to support file and system integrity.

The third section describes basic system algorithms for creation, modification, and display of the data base elements. The fourth section describes the structure of the Data Dictionary, which determines the actual format of particular data elements. The fifth section describes other data structures used by the system. The appendix lists the names of system procedures grouped by function.

FMS DATA BASE DESCRIPTION

Introduction

FMS uses the AUGMENT file system for its data base storage. The basic data item, called a "field", is represented in a character-string format. Fields are grouped together into records, each record being an AUGMENT statement. The following paragraphs describe the organization of fields and records in the entire data base.

Fields

A field is a character-string representation of a data item. Fields are delimited by left and right field delimiters. The left field delimiter is usually the concatenation of the field name (all upper case), a colon, and a space; the right field delimiter is usually an EOL character. Thus, the left field delimiter for the ENGINEER field would be "ENGINEER: ". A list of currently valid field names appears below in the Data Dictionary section of this document. The following fields have delimiters which do not follow the rule:

JON (the job order number). The left delimiter for this field is a left parenthesis as the first character of a statement, followed by the letter "J". The right delimiter is a right parenthesis.

IDENT (the user ident for manpower charges), and NUMBER (the PR number). The left delimiter for these fields is a left parenthesis as the first character of a statement. The right delimiter is a right parenthesis.

TITLE (the Effort title). The left delimiter is the first right parenthesis in the statement followed by a SPACE. The right delimiter is EOL.

The various data types recognized by FMS are represented as follows:

STRING -- Represented as text. Unless otherwise permitted, all strings are upper case.

INTEGER -- String representation of the integer.

FLOATING POINT NUMBER -- String representation of a double precision floating point number.

DATE -- String representation in the form dd-mmm-yy, where dd is one or two digits representing the day of the month, mmm are the first three characters of the month's name, and yy are the two last digits of the year.

DOLLAR -- Same representation as integer. (It differs only in the way FMS formats dollar amounts for output.)

Records

Fields are appended to each other to form records. A record is an AUGMENT statement. The FMS data base consists of the following records:

Effort

PR

Ledger-Effort

Ledger-PR

Manpower

All these records have the same internal structure. A field is added to a record by appending the left delimiter followed by the string representation of the field and then the field right delimiter to the record. Deleting a field from a record entails the deletion of the left and right field delimiters as well as the content of the field item itself.

If nothing appears between the left and right field delimiters the field is considered empty. In the ledger, an empty field designates the removal of a previously existing field.

System Files and Their Organization

Records are grouped together and stored in files. The following paragraphs describe the structure of the various files that exist in FMS.

TPO Files

TPO files contain most of the Effort and PR information. (They do not contain the manpower data; also, eventually the FMS system will interface to other data base systems.) The data base may contain an arbitrary number of TPO files.

TPO files are divided into "products". Products are divided into "groups". Groups are divided into "areas". Areas contain Efforts (which include PRs). The AUGMENT file system provides for such a structure by the use of branches. Each such branch is headed by a

statement that includes the branch title, possibly preceded by a branch name in parentheses. The name, if it exists, is also the branch name of the branch in the AUGMENT addressing scheme.

The most widely used record in the data base is the Effort. An Effort is equivalent to a contract or a project and may extend over more than one fiscal year. Efforts are either planned or actual; planned Efforts are distinguished by the letter "P" at the end of the JON. Following every Effort are PR records sorted chronologically according to the estimated start date (which is one of the PR fields). The earliest PR is called the leading PR. Every Effort must have at least one PR. The PR record contains information concerning a given fiscal year while the Effort contains general information as well as summary information about all PRs in the Effort.

Ledger Files

The system keeps a ledger of all changes made to any Effort or PR. These are stored in ledger files. Each ledger file belongs to only one TPO. However, each TPO file may have many ledger files associated with it. This is because there is much more information to be stored in the ledger and there is a limit to file size. The system automatically creates new ledger files when necessary (e.g., when the file size is larger than a safe maximum).

The ledger file consists of one ledger branch for each JON. Each ledger branch has a header consisting of a name only (enclosed in parentheses). The branch name is the JON, preceded by the letter L. (This form was determined by the fact that JONs start with a digit and AUGMENT statement names must start with a letter.) Each branch consists of Ledger-Effort records, described below.

Data is entered into the ledger with the Update command (see below) and may be retrieved by the use of the appropriate sequence generator. (See the description of the History command.) A ledger is kept for actual Efforts only. Ledger-Effort records are never modified; rather, a new Ledger-Effort record is created for every update. These records look like a regular Effort record, except that only the fields that have been modified are recorded. Note that Ledger-Effort records will have a PR only if changes to the PR were made, and then only the affected PR will be represented.

Manpower Files

The manpower files are similar in structure to the ledger files. They consist of manpower branches named with the JON preceded by

"MP". The Manpower record itself is a standard FMS record. Data is entered into the manpower file by the FORM2 subsystem (using the F2UPDATE procedure). The data may be retrieved via a special sequence generator. (See the description of the Display command and sequences below.)

Control File

The control file serves as the center of information ABOUT the data base. In the control file, the Data Base Administrator controls who may access the data base and what level of access is permitted; it also contains the actual file names and lock information.

Most statements in the control file are named to facilitate access to them. Name delimiters are always left and right parentheses. To facilitate searching by content and string matching, information should be kept in all upper case.

The following describes each branch in the control file, the meaning of its content, and the associated syntax. Some of these branches should be changed only by the system (they are "automatically" changed); others are read-only to regular users but can be modified by the Data Base Administrator.

The TPOS Branch

The TPOS branch provides the mapping between the "generic" TPO names and the actual file names associated with them. This branch may be modified only by Data Base Administrators. Each statement in this branch contains a generic name for a TPO followed by an AUGMENT link to the appropriate TPO file. Note that many generic names can point to the same file, thus allowing for synonyms. The file name portion of the link (without the directory, the extension, and the version number) is considered the "specific" name of the TPO and is used to find related files, such as ledgers.

Example:

```
DEMO <FMSDIRECTORY,TPODEMO.NLS,>
```

In this example, the generic file DEMO is represented by the actual file TPODEMO in the FMSDIRECTORY directory. The specific name is TPODEMO.

The LEDGERS Branch

This branch contains links to the various ledger files. It is maintained automatically by the system. At the first level of this branch, the specific names of the TPOs are found. Following each specific TPO name, there is a plex of links to all the associated ledger files, with the most recent first.

Example:

(TPODEMO)

<FMSDIRECTORY, TPODEMO-LEDGERFILE3.NLS,>

<FMSDIRECTORY, TPODEMO-LEDGERFILE2.NLS,>

<FMSDIRECTORY, TPODEMO-LEDGERFILE1.NLS,>

The MANPOWER Branch

This branch contains links to the various manpower files. It may be modified by Data Base Administrators. Each statement in the branch contains the specific name of the TPO followed by a link to the associated manpower file.

Example:

(TPODEMO) <FMSDIRECTORY,TPODEMO-MANPOWERFILE.NLS,>

The NUMBERS Branch

When Efforts are created, the user can supply the system with the new JON or request that the system create a unique new JON. These system-supplied JONs are generated by concatenation of the project number (4 digits), a unique three-digit number, and the letter P. The unique three-digit number is taken from the NUMBERS branch of the control file. The statements in this branch consist of the specific TPO name followed by the "current" unique number. (This number is incremented whenever a new system-generated JON is requested.) This branch is maintained automatically by the system.

Example:

(TPODEMO) 173

The ACCESS Branch

This branch contains the information concerning access to the data base. The Data Base Administrator may modify this branch. The statements in this branch consist of the user's AUGMENT ident within parentheses, followed by the FMS password and then the access level assigned to the user. All the information must be upper case.

Example:

(ROM) ROMSPASSWORD 3

The LOCKS Branch

This branch contains the various locks of the data base. It is maintained automatically by the system. The locking mechanism is described below. The syntax of the lock is as follows:

The JON of the locked Effort, preceded by a J and surrounded by parentheses.

The name of the user who has locked the Effort.

Date and time the Effort was locked.

Link to the statement in the data base where the Effort belongs. (This link includes an AUGMENT Statement Identifier (SID); hence, renumbering of SIDs is prohibited for any files that have any locks.)

A code character indicating the type of lock: M for Modify or C for Create.

Optional comment.

Example:

(J55555555) ROM 1-Jan-79 14:32 <FMSDIRECTORY,TPODEMO.NLS,09
> C This is an optional comment

The TEMPLATES Branch

This branch contains the filters that are provided by the system for displaying of data. (See the description of Display command below.) It is maintained by the Data Base Administrator. Two types of filters currently exist: built-in and recognized. The built-in filters are those that are known

to and hard-coded into the system. In addition, a filter can be added to the control file and be made accessible by all users. These filters are placed in the "recognized" branch. (The name "recognized" is due to the fact that the names of these filters are recognized by the FMS Frontend processor. A regular user cannot distinguish between the two types.) The FMS User Guide specifies how to write such filters.

Example:

```
(BUILTINS)

   (CONTRACTOR-LIST)

        !JON....  !PERFORMER.............

(RECOGNIZED)

   (VALUE)

        !JON.... !TITLE.................. !VALUE......
```

The REPORTS Branch

This branch, maintained by the Data Base Administrator, consists of two sub-branches for built-in and recognized reports (just as is the case with filters). However, instead of the report template itself, these branches contain links to the reports. (This is because report templates are lengthy and the control file should remain as small as possible.)

Example:

```
(BUILTINS)

   (JB) <FMSDIRECTORY, JBREPORTTEMPLATE.NLS, 1>

(RECOGNIZED)

   (MYSPECIAL) <ROM,SPECIALREPORT.NLS,TEST>
```

Accessing the Control File

In general, programmers do not have to access the control file directly. Primitives exist in the FMS system to provide all the necessary information. (See below for a list of these.)

Since the data base operation relies heavily on the information

in the control file, it is important that the file remain accessible to all users most of the time. The control file is accessed for write only in the following cases:

Creation of default JONs.

Locking of Efforts (for creation, modification, and deletion).

Unlocking of Efforts (in update or abort).

Work File

The work file is the user's scratch area. It is created automatically by the system when a user first enters FMS/DES and remains in his/her directory forever after. It is never deleted and will never be archived.

The information in the work file is considered private, and therefore the file is protected so that only the user to whom it belongs can access it.

The work file is used in a variety of ways and has no prescribed structure or constraints on its contents. It is the responsibility of the various operations that involve the work file to maintain the required information. Generally, whenever an FMS-related process needs temporary file space, the work file is used.

THE FMS FILE SYSTEM: SECURITY MECHANISMS

Introduction

The file system used by FMS is based on the AUGMENT structured file system. However, since FMS is primarily a data base management system, it has many special requirements centered around preserving file integrity during access and modification of system data by many users. Therefore the FMS system enhances the AUGMENT file system by providing special primitives to perform basic operations. These operations include opening, closing, and updating of files, as well as the deletion of modifications to the data base.

File System Integrity

A data base is vulnerable to unexpected and undesired modification during an actual write operation to the data base itself. Vulnerability in this case means that the computer system crashes and leaves the data base in an undefined state. To ensure the integrity of the data base, the periods when system files are vulnerable are minimized. This is accomplished by adhering to the following algorithm whenever data must be written into the data base:

1. Prepare the data to be written in the work file or identify the data to be moved in the data base. (This requires data base read access only.)

2. Identify the destination of the data in the data base. (Again, only read access is required.)

3. Reopen all the appropriate files for write. This operation is performed by the special procedure WRTOPEN. If the file is locked by someone else, another attempt to gain access will be made after a delay. If it is then still unavailable, the process will be aborted. If the file is not locked, the procedure ensures that the highest version of the file has been obtained before actual writing of data takes place. This procedure eliminates the conflicts of simultaneous write access to the files.

4. Update the actual data base files.

If more than one file is involved, steps 1 and 2 are performed for all files, then step 3 is performed for all files, and finally step 4 is performed for all files.

Following the above algorithm results in a very short vulnerable period, the duration of step 4. If only one file is involved, the data is completely safe since an old version is always available. If many files are involved, the computer may crash after some files were

successfully updated while others were not. While such situations cannot be eliminated completely, we have minimized their probability by restricting the times during which data is in the process of being written onto files in the data base. Note that before step 4 is executed, one can always use the AUGMENT "Delete Modifications" command on all relevant files to preserve file system integrity. In fact, if it ever happens that the computer crashes while writing on any of the files in the data base, the safest thing is to Delete Modifications in all files.

Locking the Data Base

Beyond the preservation of data base integrity at the file level, data base integrity must be preserved at the record level. Hence a locking mechanism is devised to ensure that no two users will simultaneously gain write access to the same record. When a user wishes to modify an Effort, the control file is checked for lock information. If the record is locked by another user, write access is denied to the requestor. If no one else has the file locked, a lock is placed in the control file and the requestor is granted the access.

Note that record locking is separate from file locking: different users can lock different Efforts in the same file. However, for safety reasons, every user may have at most one Effort locked. See the description of the lock below.

Passwords and User Security Levels

Further data base security is provided through restrictions to system access. In order to enter the system, the user must provide a password that is separate (and preferably distinct) from his/her login password. Only users who are recognized by the system can access the data.

Once identified by the system, every user is assigned access capabilities. The system currently identifies two levels: read and write. Some users may only read the data and never modify it. It would be possible to limit the use of more sensitive commands to privileged users.

SYSTEM ALGORITHMS AND OPERATION

Introduction

This section describes the basic algorithms for the various commands available in FMS/DES. It is divided into two subsections, concerned with retrieval and modifications. Whereas retrieval is done mainly in FMS and modifications to the data base are done mainly in DES, some operations (such as simulation) are common to both systems.

Retrieval

Sequences

Retrieval operations on data bases are often done on a set of records which, from the system's standpoint, are unordered (i.e., are not stored consecutively or otherwise linked together). To facilitate various data base applications, the system must provide a standard way of handling such sets, allowing the user to first define the set and then request the system to hand over the relevant records to an application procedure, one at a time. The AUGMENT sequencing mechanism utilizing sequence generators lends itself to the definition of such sets, and FMS makes extensive use of these sequence generators. To that end a few sequence generators are available in FMS and serve to generate various portions of the TPOs, the ledger, and the manpower data. These generators are described below.

The basic sequence generator allows for the generation of an arbitrary set of branches and is used to generate both the TPO data and the ledger data. A source array (see below), containing branches of records to be considered, must be constructed beforehand, and serves as the initial definition of the set. For the TPO data this source array is usually user specified, and the procedure CNVSOURCES converts the AUGMENT Frontend Processor representation of the sources into the standard FMS array format. For the ledger the procedure MKLDGARRAY assists in constructing a source array for all branches referring to a given contract (even with renamed JONs). After the source array is specified, the use of the basic sequence generator takes over the processing.

To initialize the generator, the procedure MKSRCSEQ must be called and passed the source array, two (optional) filters, and a simulation flag. The sequence handle returned should be used to identify the set when records are requested (using the procedure NEXT). The sequence generator works as follows:

Each statement identifier (STID) in the source array represents a branch, and all the statements in that branch, including the header, will be generated.

For each such branch, an auxiliary sequence is opened (one at a time) with the designated viewspecs and the first filter, if any, as a content analyzer.

Every time a record has to be generated, the auxiliary sequence generator is checked; if the sequence has not been exhausted, a record is generated and passed through the second filter. If the auxiliary sequence is exhausted, a new sequence will be opened for the next branch in the source array.

If simulation is requested, then before a statement is returned from the generator (and after it has passed the filters), the work file is checked to see whether a statement with an identical name exists there. If it does then the work file statement is returned; otherwise, the original statement is returned.

This double filtering allows screening records in two steps before they are passed for further processing. This mechanism is advantageous as it allows a quick, initial checking in the first step and a more elaborate check at the second step. A host of procedures in a *filter format* are available within FMS. Note that using the send (or sport) mechanism in the specified filters defeats further filtering and simulation. Thus, if the first filter "sends", the second filter will not be invoked. If either filter "sends", simulation cannot be performed.

The basic sequence generator provides an efficient mechanism to generate a sequence of arbitrary FMS records. Manpower data poses a different problem as it is stored in a separate data base, but in many cases the manpower data needs to be viewed in the context of the TPO data. Manpower data belonging to a given JON should be presented together with other data belonging to the same JON. A sequence generator to do just that, the MPSEQGEN, is provided. This sequence generator is similar to the standard AUGMENT INCLUDE sequence generator.

Its operation is essentially as follows: The sequence generator deploys two other generators called the primary and auxiliary. The primary sequence generator will generate all the statements in the structure as specified by the user when the sequence is opened. The primary sequence generator obeys all viewspecs except for i and O (i.e., it does not filter or invoke another sequence generator). After the last PR of an Effort is generated, the sequence generator searches for a matching manpower record; if one

is found, an auxiliary sequence generator is opened and its statements are returned until exhausted. If a matching manpower record is not found, an auxiliary sequence is not opened and statements from the primary sequence will continue to be generated. (Simulation is currently not supported on manpower data.) This sequence generator is currently used when manpower data is requested in the standard FMS Display command. It is, however, a standard mechanism to merge data from different data bases both on the same host and cross-network.

Displaying and Scrolling

Displaying and scrolling are the basic user-oriented retrieval operations (triggered by the Display and Scroll commands). These commands use the standard AUGMENT screen formatting mechanism for their operations and do not provide for any data reordering; rather, the data appears as it happens to be stored in the data base (i.e., no ordering of Efforts and PRs according to estimated start date).

The Display Command

The Display command works as follows:

The template source for the filter is identified, "compiled" (see below), and stored temporarily.

The universal filter (procedure TEMAPLY) is instituted as the current content analyzer.

If manpower data is required, the appropriate sequence generator (see above) is instituted as the current sequence generator.

Scrolling related information is reset. (The scroll stack is reset and the scroll filter set to TRUE.)

Effectively, a Jump is performed with the appropriate viewspecs.

Templates

A template, in the context of the Display command, is a simple textual description of record formatting that controls the format in which records are converted from the internal representation to a human-readable form. When a template is applied to a given record, an output text is generated according to the formatting directives of the template. Details of the syntax

of templates is given in the FMS User Guide. Since the textual representation of templates as specified by the user is quite inefficient for machine use, a compilation process is devised that converts this text into an internal data structure which we now describe.

A template may consist of several statements, each to be applied to different records of the data base. Each such statement consists of noise text which is copied verbatim into the output stream, and field designators that will be replaced by the appropriate field value when the template is applied to a record. A template can therefore be divided into "entries", each of which contains a field designation along with the noise text that precedes it (either of which can be empty). A template entry is compiled into a three-word data structure as follows:

word 1: Character count for the beginning of noise text

word 2: Character count to the end of noise text

word 3: Left half is field token (type) or procedure address and right half is field width.

The first two words (character counts), when used as second words of L10 text pointers to the source statement, will construct text pointers that delimit the noise text. The field token is the identifier used for retrieval of fields from records. (See Data Dictionary below.) If a procedure rather than a field is designated, the address is stored here. The width is the number of characters into which the field should be filled. An entry with no noise text will have the text pointers surrounding a null string (i.e., they would point to the same character). An entry without a field token or retrieval procedure will have a 0 in the third word.

The entire template data structure is as follows:

Number of (three-word) entries in this template statement.

STID of the source template (this is the first word of the text pointers)

entry 1 (3 words)
.
.
.
entry n (3 words)

Filters

The universal filter is the procedure TEMAPPLY, which serves as a content analyzer. As a content analyzer, it is called for every statement in a structure. For the current statement, it will do the following:

Identify the statement being passed as containing an Effort record, a PR record, or a manpower record.

For a PR record, will check whether the project, pec, and fiscal year match those stored in the globals globproject, globpec, globfy.

Call the procedure TEMFILL to fill the template, passing the appropriate compiled structure.

Set scrolling information. (See description of Scroll below.)

Return the formatted text using the "send" mechanism. (The content analyzer as such always returns FALSE.)

The actual filling of the template is done in the TEMFILL procedure, which receives as parameters the compiled data structure and the STID of the record from which data should be extracted. The procedure goes over all the entries in the compiled data structure and generates a string containing the noise text as specified in the template and the value of the retrieved data. Thus, the following is performed for each template entry: Two text pointers are constructed; the first word is the STID found in the beginning of the template data structure and the second words are those in word 1 and word 2 of the entry. The corresponding text is appended to the output string.

The LH of the third word is examined:

If it is zero, the current entry processing is done.

If it represents a procedure, the procedure is called. The arguments to the called procedures are: the output string (to which the procedure should append text if necessary), the STID from which data should be extracted, and the width of the field. If the procedure returns FALSE, the filling process is aborted for this statement (by returning FALSE). If the procedure returns TRUE, the processing of the entry is complete.

If it represents a field token, the field is retrieved and formatted according to the width specified in the entry and the type of the field. If the field length is greater then allocated, it is truncated from the right. Dollar amounts are right justified; all other fields are left justified. If the requested field does not exist, dashes will be inserted instead of the field value.

A template, as described earlier, may consist of various statements, each to be applied to a different record (Effort, PR, etc.). This, however, does not preclude the use of PR fields in the Effort template and vice versa. The following convention is used in such cases:

An Effort field requested in a PR template will be extracted from the "owning" Effort.

A PR field appearing in a Effort template will be extracted from the leading PR.

The Scroll Command

The Scroll command, described in the FMS User Guide, is complementary to the Display command. It moves along the data structure without changing the filter that is in effect. Associated with the Scroll command is an array used as a stack (the scroll stack). The first word is an entry count and the entries are STIDs, each representing an STID that is or was on the bottom of the screen.

The scrolling is performed by instituting a filter (the SCRLFILTER) different from the one used in the Display command. This filter changes the current statement in the sequence to the one found on top of the scroll stack and institutes the TEMAPPLY as the current content analyzer. (See Display, above.) The net result is that the statement that previously appeared on the bottom of the screen will now appear on the top.

Scrolling backwards pops statements from the scroll stack. Scrolling forward adds the new STID to the top of the stack.

Searching

Searching the data base is performed by one central procedure, CFIND. The searching algorithm is performed on a portion of the data base described by a source array. A sequence is set up to produce all the records indicated in the source array. For each

such record a value of a given field is extracted and the relation is tested. If the record passes the test, the Effort to which it belongs is copied to the work file. If the domain includes the work file itself, then all records in the work file that do not pass the test are removed from the work file.

The testing mechanism recognizes four relational operators: equal, greater than, less than, and present. Each of these can be preceded by a negation operator. Eight possible relations are provided as primitives: present, absent, greater than, not greater than (less than or equal to), less than, not less than (greater than or equal to), equal, and not equal. More elaborate relations can be performed by successive applications of primitive testing. The relations (other than absent and present) are tested according to the field type: numeric data is tested numerically, strings are tested lexicographically, and dates are tested chronologically. These tests are performed at the low level procedure TESTRELATION.

Sorting

FMS sorting uses the sorting mechanism provided by AUGMENT. This mechanism relies on a sort key, a program similar to the content analyzer. A sort key program for FMS is provided (the FMSSKEY procedure).

Two parameters govern the sorting: the type of field to sort by (the variable SRTTYPE) and an inverse flag (the variable SRTINVERSE). The AUGMENT sorting mechanism invokes the sort key for every statement being sorted. The sort key attempts to extract the given field from that statement and sorts acoording to its type: numeric fields (integer, dollar amounts, and floating point numbers) according to their numeric value, dates according to their chronological representation, and strings according to lexicographical order (case independent). If the field does not exist in the statement, it will be put at the end of the sort in no particular order. If the inverse flag is set, the sorting will be in the reverse order, except for statements which do not include the specified field -- these will be put at the end of the sort.

The procedure FMSSORT provides the user interface to the sorting of FMS data branches.

Simulation

Simulation is the operation of testing, modifying, and using the

data without actually modifying the data base. This is done in FMS by placing a copy of the simulated Effort in the work file and requesting simulation from the appropriate commands.

When data is gathered for the various reporting commands, it can be obtained from either the data base (true data) or from the work file (simulated data). The general sequence generator (see section above) has the mechanisms to perform this function, invisible to the processing routines. A user can open the sequence using that sequence generator, then set the SEQSIMULATE flag to TRUE and receive the simulated data. If the flag is set to FALSE, actual (non-simulated) data will be provided. Note that simulation is done on all records present in the work file, independent of how they got there. Hence, the Find command that places FMS records in the work file can serve as a tool identifying records for simulation.

In order to modify the data (for simulation purposes), the standard modification mechanisms are used.

Reporting

FMS makes extensive use of the general AUGMENT report generation facilities, which rely on the existence of a report template and directives for filling in that template. All reports require a domain of records to work on. These take the form of a source array, of the kind used elsewhere in FMS. A sequence to generate all the records in that domain is opened (using the general sequence generator so that simulation is available) and passed to the report-filling mechanism in the standard way (using the procedure CFILL).

The reporting facility works in a fashion similar to the display mechanism: the system distinguishes between built-in templates and those not within the system. The built-in templates are different from the others in that some of the data that goes into the report is collected from the user beforehand at the AUGMENT Frontend level, thereby reducing the communication between the user and FMS once the template filling has begun. Those templates not built-in are divided into those recognized (where the CML knows the name of the report) and those provided by users.

The AUGMENT reporting facility has a standard mechanism to extract data from data bases. FMS provides a set of procedures to interface the template-filling process with its associated data bases. These procedures enable the user to extract a field from a record, get the next record in a sequence, find Efforts in the TPO, etc.

Together with the processing available at the template-filling level, these procedures provide a general facility for users to generate arbitrarily complex reports.

The current system implements two reports in a different (hard-coded) fashion: the Sum and History command reports. The History command report provides a summary of ledger information. It is implemented on an experimental basis and hence it is hard-coded. Eventually, it will be converted to a regular report which uses the sequence generator that traces the ledger. The Sum command reports the sum of a given numeric field over a set of (possibly simulated) records. It is implemented separately to increase processing efficiency of what is a very basic operation. These two reports are discussed in greater detail in the FMS User Guide.

Modification

Field Modification

Once a record has been placed in the work file (either after a Create or Modify took place in DES or as a result of searching or simulation in FMS), the fields that comprise it can be edited. This is done by the Increment, Decrement, or Set commands, which are implemented internally through the procedures CINCFIELD and CSETFIELD.

Setting a field requires the verification of the syntax of its new contents and then the insertion of the new field. If the field previously existed, a new value replaces the old; otherwise, a new field, including the left and right field delimiters, is added to the record. The following is checked before the setting action is complete:

Attempts to set values for the JON are rejected. Only the Update commands may set a JON.

Before any of the PR dates are set (i.e., pre-initiation, initiation, commitment, and obligation), the PR record is tested to ensure that the type of buy is set.

Upon setting any of the dates mentioned above, the estimated start date is automatically set (or changed). The new estimated start date is the date being set plus an offset, which is dependent on the type of buy (specified in another field in the PR). These offsets are defined as constants in the FMS system and are based on actual experience at RADC.

Fields can also be modified by incrementing or decrementing current values. In this case, the system will evaluate the current field value, add or subtract the increment or decrement, and proceed with the regular field-setting algorithm, including the verification of the new value. Increments are meaningful only for numeric data and dates (in the latter case, the increment is in days). If a field is empty, the increment process assumes its current value to be 0. (An empty date field is undefined.)

Another way to modify a field is to remove it altogether. However, removal is not the opposite of insertion since the operation must be recorded in the ledger. The following algorithm is used for field removal:

1. Verify that the field is removable. Information concerning which fields may be removed is stored in the Data Dictionary.

2. The field is removed from the record by removing the field contents and both of its field delimiters.

3. If the field was previously present and ledger information is being constructed, an empty field (with nothing between the left and right field delimiters) is appended to the ledger record.

Record Manipulation

Mechanisms are provided to manipulate the records that make up the data base. These mechanisms enable programs to create new records, modify existing records, move records within the data base, and delete records from the data base. The following sections elaborate on these features.

Create/Modify Records

The basic mechanisms to modify the data base are the Create and Modify commands. Create is used to generate new records in the data base, while Modify is used to edit already existing records. In both cases a scratch copy of a record is put in the work file, and is then manipulated by the user. It is finally incorporated into the data base via the Update command. While most of the operations performed after issuing the Create and Modify commands are identical, some slight differences exist. The global flag MODMODE is therefore set to TRUE if an Effort is being modified and set to FALSE when it is being created. The following describe the operation of the Create and Modify commands in more detail.

The Modify Effort Command

1. The Effort to be modified is looked for in the appropriate TPO. If it is not found, the operation is aborted.

2. A lock is placed on the given Effort. An error condition will occur if the Effort is locked by a different user.

3. A copy of the Effort branch (with all its PRs) is placed in the work file. All subsequent editing will be performed on this temporary copy.

4. A temporary ledger branch is started. This branch consists (at this time) of the Effort name only. All edits made to the Effort in this session will be recorded in the temporary ledger branch that will eventually be incorporated into the permanent ledger at update time.

5. The MODMODE flag is set to TRUE.

The Create Effort Command

1. The destination is checked to ensure the area is legal. (The area is the only level under which Efforts can be placed.)

2. The TPO is searched to ensure that an Effort with the same JON does not exist. (The system thereby ensures JON uniqueness within any TPO.)

3. A temporary Effort record, consisting of the JON only, is placed in the work file. No ledger branch is necessary for Effort creation.

4. A PR is placed under the temporary Effort record. For actual Efforts, the PR name is provided by the user. For planned Efforts, the PR name is constructed internally by the system and will be "B7-0" followed by the project number followed by "P". The project number is taken from the fourth through seventh characters of the JON.

5. The MODMODE flag is set to FALSE.

After either the Create Effort or the Modify Effort command is issued, temporary records that can be edited and later updated exist in the work file.

In addition to the creation of Efforts, additional PRs can be added to an Effort that is being modified or created. This is done by simply adding another branch under the Effort being edited (and under the ledger branch if any) with only the PR number set in it.

Record Deletion

In general, record deletion, much like field removal, is not the inverse operation of creation. In some cases traces should be left in the ledger for these activities.

Delete Effort

After creation, actual Efforts have an associated ledger branch in one of the ledger files. By definition ledgers cannot be modified in any way; once placed in it, the information cannot be removed. Therefore, an actual Effort can-

not be deleted. Only planned Efforts, which have no ledger associated with them, can be deleted. In this case the system will:

1. Ensure that the Effort is not locked by any other user.

2. Lock the Effort for the given user (to avoid any attempts from others to access this Effort while it is being deleted).

3. Delete the Effort branch from the data base file and update the file.

4. Remove the lock from the control file. (This lock is meaningless by now since the Effort does not exist any more.)

Delete PR

A PR can not actually be deleted; however, its operation can be cancelled. Since traces of that PR exist in the ledger, the cancellation is accomplished by setting the obligated amount to 0 (thereby effectively cancelling it) and deleting the PR record from the Effort. It is a requirement that every Effort have at least one PR. While this is not checked at this point (i.e., the last PR may actually be cancelled), it is checked at update time.

Moving Records

Records sometimes need to be moved from one data base location to another and eventually out of the data base for final archival. The Move and Finish commands provide these mechanisms.

Move Effort

Moving an Effort results in moving the Effort record and all its PRs from one data base location to another. Since Efforts have corresponding ledger information, this information may need to move as well. The following files are involved in Effort movement: the source TPO file where the Effort currently resides, the destination TPO file into which the Effort is to be moved (which may or may not be identical to the source TPO file), the source ledger files where the ledger branches for the Effort currently reside, and the destination ledger file. (Note that the source led-

ger branches may reside in more than one file since there may be more than one ledger file per TPO and the Effort might have changed the JON in its lifetime.)

Effort movement proceeds as follows:

1. The source is identified as a valid Effort record.

2. The destination is verified to be a legal area.

3. The Effort is locked. This involves checking that no other user is currently modifying this Effort and that an Effort with the same name is not being created in the destination TPO.

4. All ledger branches, if any, are identified. If the Effort has changed its JON, this involves all ledger branches for the previous JONs as well (which may reside in more than one ledger file).

5. All involved files are opened for write access (thereby prohibiting access to them while the operation is in effect).

6. The Effort branch is moved (in the AUGMENT sense) from source to destination. Then ALL ledger branches are moved to the destination ledger file.

7. All involved files are updated (in the AUGMENT sense).

8. The lock is removed.

Note that moving a planned Effort (that does not have a ledger) or moving any Effort within the same TPO does not require any ledger information movements.

Move PR

A PR can be moved from one Effort to another. However, to avoid any associated ledger problems PR movements are allowed only between planned Efforts that do not have any ledger information.

PR movement proceeds as follows:

1. Source and destination Efforts should not be locked.

2. Source and destination Efforts must be planned.

3. The PR should not be the only PR of the source Effort (so that at least one PR remains at the source after the moving).

4. The destination Effort is checked to ensure that it does not include a PR with the same number. (PR numbers are unique at least within an Effort.)

5. The destination Effort is locked.

6. The PR record is moved (in the AUGMENT sense) from source to destination.

7. All files are updated (in the AUGMENT sense).

8. The lock is removed.

Finish

The data base, as described so far, will be ever increasing in size. However, most of the information is time dependent and becomes useless after a while. For example, after a contract is completed and related activities satisfied, the information about the contract is no longer needed, at least not in an immediate online fashion. The Finish command provides a mechanism with which data can be removed from the data base and archived.

The Finish command will assemble all information relevant to an Effort: the current Effort record with all its PRs and all the ledger branches that were ever associated with this Effort (even if the JONs were changed). All this information will be placed in a newly created file that will be marked for archival. The original data will be permanently removed from the data base.

The Finish algorithm is as follows:

1. The Effort is locked to ensure that nobody else is currently modifying this Effort.

2. A new file is created in the standard FMS directory whose name is Jxxxxxxx-FINISHED, where xxxxxxx stands for the current JON of the Effort being finished.

3. The Effort branch is copied to the newly created file.

4. All associated ledger branches are identified and copied into the new file following the Effort data (and at the same AUGMENT level).

5. The new file is updated (in the AUGMENT sense).

6. The file protection is set to 424200B, which will allow group members to read this file and allow nobody to write on it. Thus, the file is frozen. In addition the file is marked for immediate archival.

7. All the associated information (TPO and ledger data) is now deleted from the data base.

8. All associated files (TPO and ledgers) are updated (in the AUGMENT sense).

9. The locking information is removed.

Note that there does not currently exist a mechanism to systematically retrieve the file from the archive and to use the data stored in it.

Update

Once all modifications to a given record are complete, they must be incorporated into the data base. This is done by the Update command and entails copying or replacing data into the TPO data file and may require copying data into the ledger files. In the process of updating, the JON of an Effort can be changed. (This is the only place where one can change the JON; see remarks about the Set command.)

Update Without Renaming

In general, the following occurs when an Effort is updated (without renaming):

The Effort data is put in the TPO file. In the Create mode the data is inserted, while in the Modify mode the old Effort is replaced by the new data.

If the Effort is an actual one, the ledger must also be updated. In the Create mode the Effort data will simply be copied into the ledger in the appropriate new branch. In the Modify mode the ledger information in the work file will be copied into the ledger. The information put in the ledger will include a MODIFIED-BY entry that contains the iden-

tifier of the user and the time of update. This information is inserted automatically by the system without user control. In addition, a REASON entry is inserted that identifies the reason for the current update. This field is user controlled.

The following is the detailed algorithm:

1. Verify the current Effort in the work file.

2. Verify the lock information and follow the link contained in the lock. (This link points to where the Effort belongs in the TPO data file.)

3. If ledger activity will be required, check the ledger files. In the Modify mode the ledger branch must exist, and in the Create mode the ledger branch must not exist and will be created. Prepare the additional information needed for the ledger.

4. All relevant files (the TPO file and the ledger file, if any) are reopened for write (locked in the AUGMENT sense).

5. Data is transferred from the work file into the TPO file and the ledger files.

6. All files are updated (in the AUGMENT sense).

7. The lock is removed and the work file cleared. (See description of the Abort command below.)

Update With Renaming

Updating an Effort with a JON change is an operation that requires updating information regarding the old JON and introducing new information about the new JON. In addition to the regular update operations, the following occurs upon renaming an Effort:

An entry is inserted in the ledger of the old JON (in the REASON field) indicating that the JON has changed and the EXCEPTION field will contain the new JON. (This operation is skipped if the original Effort does not have a ledger.)

The information that usually goes to the ledger will go to the ledger of the new JON. This information will include the PREVIOUS-JON field with the old JON in it. (This step is skipped if the new Effort does not need a ledger.)

The Update Rename operation is a lengthy one due to the relatively large amounts of data to be transferred and the number of files involved. In order to secure the data in the work file, both the Effort and the ledger are re-copied into new branches in the work file. This is done because we do need to change the JON field that is also recorded in the lock (in the control file). Thus, if the machine somehow crashes in the middle of the process, the lock and the work file are still synchronized and no data is lost (though the user will have to reissue the Update Rename command).

The algorithm for Update Rename is as follows:

1. Verify the content of the work file.

2. Verify the lock information and follow the link contained in the lock. (This link points to where the Effort belongs in the TPO data file.)

3. Make copies (in the work file) of the Effort and ledger branches. (These will be the data the procedure works on.)

4. If the old JON had a ledger (i.e., in Modify mode for an actual Effort), prepare the ledger entry for the old JON. This entry will include the name of the old JON, a MODIFIED-BY entry that includes the ident of the user and the time of change, and a REASON entry that says "JON CHANGED TO" followed by the new JON.

5. The TPO file is checked to see that the new JON does not exist and that an Effort with that JON is not locked. This ensures that the new JON is still unique within the TPO file and that another user cannot currently create an Effort with that JON.

6. The JON field in both the Effort and the ledger branches in the work file are changed to the new JON. In addition, if a ledger is being created, a PREVIOUS-JON entry in the ledger branch is inserted containing the old JON.

7. All destination files are identified (the TPO file and all destination ledger files) and opened for write access (locking in the AUGMENT sense).

8. All information is inserted into the appropriate files.

9. All files are updated (in the AUGMENT sense).

10. The lock is removed and the work file cleared. (See the description of the Abort command below.)

Abortion and Validation

Two general mechanisms are provided both in FMS and DES. These are the Abort and Validate functions.

Abort

The Abort operation results in giving the user a clean work file. It deletes all the contents of the work file by deleting modifications (in the AUGMENT sense). In addition, the control file is checked to see if the user has any Efforts locked and releases these Efforts if so.

Validate

The Validate command allows the user to check the consistency of the information in the Efforts residing in the work file. The system requires validation of the Effort before the Update operation is performed. Validation is also useful in FMS simulation to ensure that the suimulated data is consistent.

The Validate command will inform the user of warnings as well as errors. Errors will prohibit further updates; warnings are for information only. The validation process will not stop at the first inconsistency but rather will continue as far as it can. The following is checked by the Validate command:

All the necessary fields in the Effort record are checked. These are: JON, title, value, and duration.

All necessary fields are checked for each PR. These are: number, project, fiscal-year, type of buy, and (any) amount.

The following fields must appear in pairs for each PR: initiation date and amount, commitment date and amount, and obligation date and amount.

Estimated start date for each PR must be within the scope of the contract.

Obligation date of each PR (if it exists) must be within 10 days of the estimated start date (warning).

The sum of the amounts in all PRs of an Effort must not exceed the the contract face value (warning).

The sum of the man-hours in all PRs of an Effort must not exceed that of the Effort (warning).

The leading PR must have a type of buy: A, B, E, H, I, or K.

PRs are sorted according to estimated start date. (The validation process will re-sort the PRs if necessary.)

DATA DICTIONARY

Introduction

In FMS, a dictionary contains all the knowledge about fields. This information includes attributes, syntax, and retrieval algorithms. The dictionary is a collection of tables along with a set of procedures, as described below. The dictionary is made of three principal tables. These tables are precompiled and used in a read-only mode:

"fieldnames" table. This provides the mapping between the field token and the official field name (the one contained in the left field delimiter).

"fldattributes" table. This contains the attribute information of all fields.

"fldveproc" table. This provides (indirect) information on the syntax of the physical fields and the algorithm for evaluation of logical fields.

The contents of these tables and the fashion in which they are used are described in the following sections.

Field Attributes

Every FMS field has a set of attributes associated with it. The following exist:

Exist/notexist. Since tables need not be compacted (i.e., can have holes), this is a means of finding whether a given token actually refers to a field.

Physical/logical. Physical fields are those that appear directly in the data base. Logical fields are those values that can be derived from other data physical fields.

Owning record. An FMS field can belong to an Effort record, a PR record, or a manpower record. A field cannot belong to more than one record.

Data type. A field can be of type: string (text), integer, floating point (decimal), dollar amount, and date.

Shift case. Indicates whether the field contents must be all upper case.

Removable/nonremovable. Some fields may not be removed from the owning record. (See the Remove command above.)

The attributes structure is defined in the FMS system as a record (the ATRECORD) and refers to the information stored in the FLDATTRIBUTE table. However, users need not access the table directly because appropriate procedures exist to check all the various attributes.

Field Syntax

All physical fields have a prescribed syntax associated with them. The system provides mechanisms to verify the syntax of field content to ensure that only syntactically legal fields are recorded in the data base. The syntax for every physical field is verified by calling a procedure whose address is specified in the FLDVEPROC. The user need never access this table directly; the two procedures VERIFY and TFVERIFY provide an interface to these tables.

In the following paragraphs, the syntax for each field is described. Most of the fields must undergo only a very general verification, while some need very specific syntax verification. In the general case the following is checked:

A field must be a string of at least one character and at most 100.

The field may not contain EOLs.

Upper case is enforced if required by the field attributes.

Leading and trailing spaces are removed.

Dollar sign is removed (if it exists) from fields that represent dollar amounts.

Date fields are checked to ensure that they represent a legal date (e.g., 29-feb-79 is an illegal date while 29-feb-80 is legal).

Floating point numbers are checked.

Numeric fields are checked to contain digits only. (Commas, if present, are legal but will be removed.)

The following fields have specific syntax checks in addition to the general verification described above:

JON: 8 letters or digits, possibly preceded by a "J" (which will be removed).

Title: 25 characters maximum.

Duration: 1 or 2 digits only.

Manyears: 1-5 digits with one optional decimal place.

Priority: 1 to 3 digits.

Contract: acceptable syntax (described in content-analyzer syntax) is:

```
UL 5LD '- 2LD '- UL '- 4LD
UL 5LD '- 2D  '- D  '- 4D
5UL    '- 2D  '- 4D
```

Tech: 1 to 3 digits.

Task: 2 letters or digits.

Type: A, B, C, E, F, G, H, I, K.

Line: 1 to 3 digits.

Number: acceptable syntax is (L stands for any letter, D for any digit, - for a dash, and T for the letter 'T'):

L-D-DDDDL, L-D-DDDD, L-DDDDDL, L-DDDDD, L-T-DDDDL, L-T-DDDD, L-TDDDDL, L-TDDDD, LD-DDDDL, LD-DDDD, LDDDDDL, LDDDDD, LT-DDDDL, LT-DDDD, LTDDDDL, LTDDDD

Project: 4 letters or digits.

Fiscal year: 2 digits or "7T".

Der (Daily Expenditure Rate): numbers separated by semicolon and spaces (which will be removed)

Logical Field Extraction

Logical fields are those that are not explicitly stored in a record but can be evaluated based on the physical fields of the record. An algorithm is therefore associated with each logical field that defines the way it is computed. Internally, there is a set of "evaluation procedures", one for each logical field. Each of these procedures accepts as an argument the STID of the record in question and returns, in a string format, the value of the logical field. The addresses of these procedures are kept in the FLDVEPROC table. Users do not have to access this table directly; the low-level field retrieval procedures (such as GETFIELD) provide the appropriate mechanisms. Hence, the distinction (at retrieval time) between physical and logical fields is completely transparent.

In principle, if a logical field cannot be evaluated, it is considered an empty field. However, since most of the logical fields rely on more than one physical field, it may be useful for the user to know why a logical field cannot be evaluated. The evaluation procedure will therefore generate a signal (of type SIGLOGICAL) along with an error string explaining the nature of the problem.

The following is the description of the various algorithms that are used to evaluate logical fields:

AMOUNT: This is the most advanced amount information available, i.e., obligated, committed, initiated, pre-initiated. (Obligated is the most advanced.)

FISCAL-YEAR: This is the fiscal year in which the estimated start date falls.

PROGRESS: This is the procurement state, i.e., obligated, committed, initiated, and pre-initiated.

PROJECTION: This is the expenditure projections of a given PR. For the type of buy C, E, F, G, H, K, this is the most advanced amount (see AMOUNT above). For type of buy I, the projection is zero. For type of buy A or B, the projection is essentially the product of the daily expenditure rate and the length of the PR in days. If more than one PR exists, the daily expenditure rate is computed by using the the DER field (see below). The projection amount is always limited by the VALUE of the contract.

DER: This is the Daily Expenditure Rate and is computed by dividing the PR amount by its duration in days. The PR amount is computed as in AMOUNT above. The duration of the PR is the number

of days between the estimated start date (or obligation date if obligated) and the end of fiscal year or the end of contract, whichever comes first.

LENGTH: This is the length of the PR in days. It is the number of days between the estimated start day and the end of fiscal year or end of contract, whichever comes first.

SOWDUE: Computes the SOW due date. It is the Effort start offset by a given amount based on the type of buy of the leading PR.

DURDAYS: Duration of contract in days. This is the product of the duration of the contract in months and 30.44 (which is the average days per month).

START: The Effort start day. This is the estimated start day of the leading PR.

TERMINATION: This is the contract's termination day. It is the start date plus the duration of the contract (START+DURDAYS).

EFFWRITEUP: This is a one-statement description of the contract. Used mainly for reporting. The Effort writeup may be found in a special Effort writeup file that is outside the FMS data base.

Field Descriptions

Introduction

The following table describes valid fields in FMS; information in the table is arranged in columns. In the OWN column, we designate in what record the field appears (EF for Effort, PR for PR, MP for manpower). In the P/L column, P indicates a Physical field and L a Logical. In the TYPE column, we indicate the data type (S for string, I for integer, F for floating point, $ for dollar amounts, and D for date). In the Notes column, we designate I for internal (i.e., not user-settable), NR for nonremovable field, and LC if lower case is allowed.

Token	Official Name	OWN	P/L	TYPE	Notes	SYNONYM
1	JON	EF	P	S	NR	
2	TITLE	EF	P	S	LC,NR	
3	VALUE	EF	P	$	NR	
4	ENGINEER	EF	P	S		
5	PRIORITY	EF	P	I		PRI
6	DURATION	EF	P	I	NR	DUR
7	MAN-YEARS	EF	P	F		MANYEARS
8	CONTRACT	EF	P	S		
9	PERFORMER	EF	P	S		
10	TECH	EF	P	S		
11	TER	EF	P	S		
12	PAID	EF	P	$		
13	IN-PROGRESS	EF	P	$		INPROGRESS
14	UNLIQUIDATED	EF	P	$		
15	CORE-TEMP	EF	P	S		CORETEMP
16	ALT-TASK	EF	P	S		ALTTASK

17	SECTION	EF	P	S		
18	MANPOWER-LINK	EF	P	S		MANPOWERLINK
19	WRITEUP-LINK	EF	P	S		WRITEUPLINK
20	BUYER	EF	P	S		
21	EXCEPTION	EF	P	S		
22	ACD	EF	P	S		
23	RC-CC	EF	P	S		RCCC
24	ACCESSION	EF	P	S		
25	COSATI	EF	P	S		
26	REASON	EF	P	S	LC,I	
27	PREVIOUS-JON	EF	P	S	I	
28	MODIFIED-BY	EF	P	S	I	MODIFIEDBY
36	SOWDUE	EF	L	D		
37	DURDAYS	EF	L	I		
38	START	EF	L	D		
39	TERMINATION	EF	L	D		TERM
40	NEWSTARTS	EF	L	D		
41	EFFORT-WRITEUP	EF	L	S		EFFORTWRITEUP
42	MPYTD	EF	L	F		
43	MPPREV	EF	L	F		
44	MPTOTAL	EF	L	F		
51	NUMBER	PR	P	S	NR	
52	PROJECT	PR	P	S	NR	PRJ
53	TASK	PR	P	S		

54	FY	PR	P	S	NR	
55	ESTIMATED	PR	P	D		
56	TYPE	PR	P	S	NR	T
57	PREDATE	PR	P	D	NR	
58	PREAMOUNT	PR	P	$	NR	
59	INITDATE	PR	P	D		
60	INITAMOUNT	PR	P	$		
61	COMDATE	PR	P	D		
62	COMAMOUNT	PR	P	$		
63	OBLDATE	PR	P	D		
64	OBLAMOUNT	PR	P	$		
65	PEC	PR	P	S		
66	LINE	PR	P	S		
67	SOURCE	PR	P	S		SRC
68	USER	PR	P	S		
69	LEAD	PR	P	S		
70	BPAC	PR	P	S		
71	ACCESSION	PR	P	S		
72	COSATI	PR	P	S		
74	BUYING-AGENCY	PR	P	S		BUYINGAGENCY
75	TPO-LINK	PR	P	S		TPOLINK
76	PR-EXCEPTION	PR	P	S		PREXCEPTION
77	MER	PR	P	F		
78	PR-TITLE	PR	P	S	LC	

79	PRMANHOURS	PR	P	F	
85	AMOUNT	PR	L	$	
86	FISCAL-YEAR	PR	L	S	FISCALYEAR
87	PLANNED	PR	L	S	
88	ACTUAL	PR	L	S	
89	PROGRESS	PR	L	S	
90	PROJECTIONS	PR	L	$	
91	DER	PR	L	F	
92	LENGTH	PR	L	I	
101	IDENT	MP	P	S	
102	PERIOD	MP	P	D	
103	CHARGE	MP	P	F	
104	YTD	MP	P	F	
105	PREVYEARS	MP	P	F	

MISCELLANEOUS DATA STRUCTURES

The following data structures are used in various parts of FMS and DES.

Field Token

This is an integer that is being used to designate an FMS field type. It is in fact an entry point into all the tables in the dictionary. Thus, when the field token is known, the name, attributes, and syntax can be extracted. The procedure FLDCONVERT converts field names (both official and synonyms) to field tokens.

Tpoindex

This is a shorthand used to transfer TPO file pointers. It is used mostly before a TPO has been accessed (such as from the CLI to the BE). In other cases, an STID to the file is used. The tpoindex is an index into the plex of statements in the TPOS branch of the control file. For example, tpoindex=2 refers to the TPO file that is pointed to by the link that appears in the second statement in the TPOS branch of the control file. Note that two different TPO indices can actually point to the same TPO file.

Arrays

Arrays are represented in the FMS system as a word count followed by the appropriate number of words. The most commonly used array is the source array. This array includes STIDs of different branches in the data base. The sequence generators usually use array information to generate the appropriate records sequence. Of particular interest is the transfer of the source array from the Frontend to the Backend. The information is collected in the Frontend using an FEFUNCTION which packs the sources into a list. Each entry in the list corresponds to a branch. The CNVSOURCES procedure in the Backend will convert this appropriately to the standard array format.

PR Handle

In the process of modifying PRs it is likely that the same PR will be referenced over and over again. Instead of searching for the PR record in the work file by name (and possibly searching for its work file ledger branch too), a PR handle is used. The PR handle can take one of the following forms:

A zero: designates the last referenced PR. (The appropriate STIDs are stored in the globals WPRSTID and LPRSTID.)

A list whose first element is zero: Element 2 will contain the STID for the PR record in the work file and Element 3 of that list will contain the PR ledger record STID in the work file (if any).

Anything else: is a simple CML literal selection.

The procedure CNVPRHANDLE converts a PR handle to a pair of STIDs. The procedure MKPRHANDLE converts a pair of STIDs to a PR handle.

Lock Structure and Lockstid

Locks are stored in the work file according to the syntax described above. In order to pass and parse lock information, locks can be represented in a data structure similar to the one used by AUGMENT for parsed links. A lock is generally represented by an STID that points to the appropriate statement in the work file. After being passed to the LOCKPARSE routine, a data structure is returned that contains text pointers that delimit the fields of the lock: JON, ident, date, time, link, mode, and comment. A variety of procedures exist to manipulate the various lock fields.

APPENDIX: User Procedures and Variables

Introduction

This is a list of all FMS user-level procedures. Only very short descriptions are given here. Refer to the source code for more detailed information. The procedure and variable names have been grouped according to function.

Command execution routines. These are the top-level procedures that execute the appropriate commands. They are usually meant to be called from the AUGMENT Frontend only. They call the internal support routines. If you want to trace a command execution, start at these procedures.

Command core routines. This is where commands are actually executed. There is almost a one-to-one correspondence with commands. They are meant to be user-level callable, i.e., Frontend independent.

File system operations. These procedures support moving and searching within files of the data base, getting information about files (names, inversion number, etc.), and providing special interfaces to the AUGMENT file system when required.

Data base checking (fields and files). This set of procedures tests various aspects of the data base, including attributes of fields and proper ies of records and files.

Field manipulation. This set of procedures provides all that is needed for the user to extract or modify fields of records in the data base.

Data conversion. This set of procedures provides for data conversion between different data types (e.g., dates to strings) and different data structures (such as sources and arrays).

Utility. Here we have grouped procedures that might be of use to a system programmer and that are of a general nature. The various FMS operations rely on these procedures.

Arrays, sequence generators, filters. Here we present the set of procedures that correspond to the filters, sequences, and sort keys used by FMS.

Locks. This set of procedures performs the lock-related services: setting, deleting, and parsing lock information.

Data. We have included here the names of global variables that

may be of use when interfacing to other standard procedures. This list is by no means complete. The Data file contains a complete list. Most of these variables are read only. Users should double check before they program anything that changes any of the global variables.

Command execution routines

Abort:	xxabort
Cancel PR:	xpcancel
Copy:	xfmscopy
Create Effort:	xecreate
Create PR:	xpcreate
Delete Effort:	xedelete
Display:	xdisplay
Find:	xfind
Finish Effort:	*xefinish*
Generate:	xgenreport
History:	xhistory
Increment/Decrement:	xxincfield xincall
Initialization:	cmninit, desinit, fmsinit
Modify Effort:	xemodify
Move Effort:	xemove
Remove Field:	xremfield
Reset:	xfreset
Scroll:	xscroll
Set Field:	xxsetfield
Show Status:	xstatus
Show:	xfshow

Simulate Effort: xsimulate

Sort Effort: xeffsort

Sum: xsum

Termination: xxfmsterm

Update: xwfupdate

Validate: xvalidate

Command core routines

Display: cdisplay

Finish Effort: cefinish

Find: cfind

Generate: cgenreport

Increment/Decrement: cincfield

Remove: cremfield

Set: csetfield

Update: cwfupdate

File system operations

fileversion - gets the file version number given STID

findcontent - finds the next occurrence of content

findeffort - finds an Effort based on several criteria

findfbcontent - finds the content in branch

findname - looks up a (FMS) statement name in a data file

findpr - looks up a PR number in a branch

fmsdelmod - deletes modifications to data file

fmsopen - opens an FMS file

fmsupdate - updates and closes a file

getfilename - gets a data file name

gethiversion - gets the highest version of a file

getledger - finds or creates a ledger branch

getmpfile - finds an MP file given Effort file

getmprecord - finds an MP record

gettpostid - converts TPO index to STID

cnvtpo - converts TPO name to index

gtversion - gets the version number given jfn

leadpr - evaluates the leading PR

opencontrol - opens control file

stdinbranch - checks whether an STID is in branch

wrtopen - re-opens an FMS file for write

Data base checking (fields and files)

checkfield - checks whether a statement contains a field

checkpr - checks the project, pec, and fy of a PR

checkwf - checks contents of work file

chkpair - checks a pair of fields

chkprfilter - filters to checks the project, pec, and fy

haspr - checks whether an EFFORT has a PR of a given kind

isactual - TRUE if statement is an actual EFFORT

isarea - TRUE if statement is an AREA head

isdate - TRUE for fields of type date

isdollar - TRUE for fields of type dollar

isefffield - TRUE for Effort fields

iseffort - TRUE if statement is an EFFORT

isfield - TRUE if type is a field

isfloat - TRUE for fields of type floating point

isgroup - TRUE if statement is a GROUP head

isinhouse - TRUE if statement is an in-house EFFORT

isinteger - TRUE for fields of type integer

isleffffield - TRUE for logical EFFORT fields

islogical - TRUE for logical fields

islprfield - TRUE for logical PR fields

ismp - TRUE if statement is a MP

ismpfield - TRUE for MP fields

isninhouse - TRUE if statement is NOT an in-house EFFORT

isnonremovable - TRUE for nonremovable fields

ispefffield - TRUE for physical EFFORT fields

isphysical - TRUE for physical fields

isplanned - TRUE if statement is a planned EFFORT

ispprfield - TRUE for physical PR fields

ispr - TRUE if statement is a PR

isprfield - TRUE for PR fields

isproduct - TRUE if statement is a PRODUCT head

matchfield - checks whether a field in a record equals a value

Field manipulation

getfield - extracts a field from a record in string format

getlogical - computes a logical field

getvalue - extracts a field value according to data type

putfield - puts a field into a record

setfield - sets a field in a record

setpointers - sets pointers around a field

Data conversion

cnverr - calls err with conversion message

cnvmsg - prepares a conversion message

cnvprhandle - converts PR handle

convert - converts a CML record into an L10 string

convfi - converts a floating point number to an integer

convfs - converts a floating point number to a string

convif - converts an integer to a floating point number

convsf - converts a string to a floating point number

date - converts a string to a TENEX date integer

fldconvert - converts a field name to a field type

fyconvert - converts a date into a 2-digit fiscal year

mydate - converts a TENEX date integer to a string

Utility procedures

cleanwf - cleans out the work file

cmninit - initialization procedure

fmssort - FMS sort core routine

fydays - computes the number of days in a fiscal year

fydistance - computes distance a fy is from a base year

fyend - computes the last day of the given fiscal year

fynext - computes the next fiscal year from the given one

jumpto - jumps to a statement

makejon - creates a job-order-number from a data file

makelink - builds a link to an STID

makepr - creates a new PR

maketest - makes a test on Effort

mkprhandle - makes a PR handle

planstr - makes a planned PR number

prcreate - creates PR in work file

prdays - computes length in days for a PR given start day

tempcmpl - compiles a display template internally

testrelation - tests a relation between field and value

verify - verifies that a value is legal for a field

Arrays, sequence generators, filters

fmsskey - general sort key

scrlfilter - scroll command filter

temapply - filter applying a template to a statement

cnvsources - converts entries into sources array

mksrcseq - opens a sequence for sources array

srccheck - checks source arrray consistency

mkldgarray - makes an array of ledger branches

Locking related procedures

checklock - checks for lock

gtlkcomment - extracts the value of COMMENT from lock

gtlkdate - extracts the value of DATE from lock

gtlkdtm - extracts the value of DATE and TIME from lock

gtlkident - extracts the value of IDENT from lock

gtlkjon - extracts the value of JON from lock

gtlklink - extracts the value of LINK from lock

gtlkmode - extracts the value of MODE from lock

gtlktime - extracts the value of TIME from lock

gtlkvalue - extracts a value from lock

lockeffort - locks an Effort

lockparse - parses a lock

makelock - makes a lock entry

unlckeffort - unlocks an Effort

Data (variables and constants)

Miscellaneous

wforigin - origin statement of work file

ctrlstid - origin statement of control file

ltpoindex - last tpoindex referenced

maxtpoindex - maximum TPO index

modmode - TRUE in Modify mode, FALSE in Create mode

globstring - used for global string values

globproject - current PROJECT requested

globpec - current PEC requested

globfy - current FY requested

siglogical - logical fields signals

maxfld - index of highest field

Access levels

fmsaccess, desaccess, superman

Data base statement levels

prdctlevel, grplevel, arealevel, efrtlevel, prlevel

Ledger searching modes

ldgfind, ldgany, ldgcreate

Lock-related constants

Lock components

lkjon, lkident, lkdate, lktime, lkdtm (date and time), lklink, lkmode, lkcomment

Offsets into the lock data structure

lkjs - jon start

lkje - jon end

lkis - ident start

lkie - ident end

lkds - date start

lkde - date end

lkts - time start

lkte - time end

lkls - link start

lkle - link end

lkms - mode start

lkme - mode end

lkcs - comment start

lkce - comment end

Control file branches

cbaccess - ACCESS

cbbuiltins - BUILTINS

cbledgers - LEDGERS

cblocks - LOCKS

cbmanpower - MANPOWER

cbnumbers - NUMBERS

cbrecognized - RECOGNIZED

cbreports - REPORTS

cbtemplates - TEMPLATES

cbtpos - TPOS

Filter and report types

cwbuiltin, cwrecognized, cwat, cwtypin, cwfirst

Relation codes

cwpresent, cwabsent, cwequal, cwgreater, cwless

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.